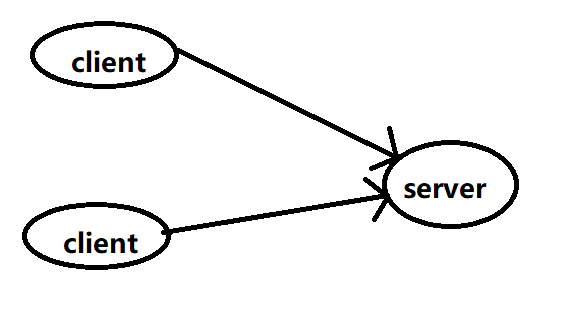
第22次课 多路复用的笔记



如果是阻塞的套接字，服务器在主线程里是没有办法同时处理多个客户端的连接和数据的收发，只有通过多线程或者多进程的方式来做到这个并发。

如果是单纯的非阻塞的套接字，服务器的CPU会一直处于一种空转的状态。

这两种编程方式都不能处理多个客户端的并发情况。

有没有办法在一个线程或者一个进程的情况下处理多个客户端的并发情况？？？

## 什么是多路复用？

处理多个文件描述符。

select：

poll：

epoll：

不同的平台有不同的多路复用技术，kqueue，

三个东西有什么不一样？

select怎么用的？

int select(int nfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout);

调用select的时候，整个线程会被阻塞起来，

readfds是它文件描述符的容器，管理了很多的socket，那nfds就是其中socket的句柄数最大的那个。

void FD\_SET(int fd, fd\_set \*set);

添加fd到set里。

void FD\_CLR(int fd, fd\_set \*set);

从set里删除fd

void FD\_ZERO(fd\_set \*set);

初始化这个set

int FD\_ISSET(int fd, fd\_set \*set);

判断这个fd在set里不是有某个关的事件发生了。

## poll

int poll(struct pollfd \*fds, nfds\_t nfds, int timeout)

使用select和poll到底有什么缺点：

1. select的不足：

1）select的实现，在内核里它是被定义成一个数组的。数组肯定就有大小，数组的大小被默认定义了1024，默认的情况下只能支持1024个socket的管理。

2）select返回后，应用层需要不断轮询这个fd\_set，去判断socket期望的是否发生。

3）select返回后，会把内核管理的这个fd\_set清空，依然需要把用户态的socket拷贝到内核的select管理的fd\_set，发生频繁从用户态的数据copy到内核里面。

poll的实现，对select进行了改进，就是用链表去保存了socket。也就是说克服了select对socket数量的限制。

高并发的情况下使用select或者poll能支持的并发是有限的。🡺 C10K(一万个左右的并发)

因此内核为了解决这个问题，内核重新设计的多路并发，也就是使用epoll来管理了socket。

## epoll

int epfd = epoll\_create(int size);

int epfd = epoll\_create1(int flags);

返回值也是一个文件句柄，

int epoll\_ctl(int epfd, int op, int fd, struct epoll\_event \*event);

如果op为EPOLL\_CTL\_ADD，则是把fd添加到epoll里。

如果op为EPOLL\_CTL\_DEL，则是把fd从epoll里删除。

如果op为EPOLL\_CTL\_MOD，则是修改fd在epoll里的事件值。

struct epoll\_event {

uint32\_t events; /\* Epoll events \*/

epoll\_data\_t data; /\* User data variable \*/

};

typedef union epoll\_data {

void \*ptr;

int fd;

uint32\_t u32;

uint64\_t u64;

} epoll\_data\_t;

int cnt = epoll\_wait(int epfd, struct epoll\_event \*events, int maxevents, int timeout);

调用了epoll\_wait, 直到epoll管理的socket有相关的事件发生的时候，epoll\_wait才会返回。

返回后，所有的事件都存储在events里面，events的size是epoll\_wait的返回值。

else if (events[i].data.fd == sfd)，如果是服务端监听的socket返回了，说明有客户端连接进来了，此时需要调用accpt去接受连接，并且accpt返回-1且errno是EAGAIN或者EWOULDBLOCK则代表需要继续处理，并不是一种异常。

accpt之后，需要调用epoll\_ctl的ADD方法把客户端的socket添加到epoll里。

else

就是客户端发送数据过来了，如果read返回-1且errno为EGAIN或者EWOULDBLOCK则继续处理，不是一种异常，如果read返回0，代表客户端的socket关闭了连接，此时服务器需要close（fd）；如果read返回值大于0，则正确地接收到了数据，此时需要process、send。

如果服务端在write，客户端close，会有什么情况：

TCP的四次挥手：

客户端发送FIN包，只是说客户端不再能够发送任何应用层的数据，客户端收到ACK(1)就确认了不能发送数据了。

任何的可靠不能丢给第三方去做，即使这是内核。

REQ --> RSP

多个线程同时调用scoket去发送数据，会出现什么情况？

*T1：buf = 123456*

*T2：buf = abcdefg*

*此时此刻，123abc4de56fg？*

socket它是一个线程安全的设计。

epoll的设计：

epoll\_wait返回的时候，只是把已经收到事件的socket返回（events），没有收到事件的socket并没有返回。克服了select的第二个缺陷。

计算机设计思想：提高效率的方法之一，就是把集中式变成分布式。

epoll把select的集中式操作变成了分布式（分散）操作，

epoll是怎么设计：

采用了红黑树管理了epoll里的所有的socket句柄，O（lgn），事件发生了，实际上是有文件系统设置的。epoll会把相应的socket添加到一个双向链表中，同时让epoll\_wait返回，此时epoll\_wait的event指向了这个双向链表。

每个线程都可以调用epoll\_wait /

内核文件系统，内核文件系统如果需要设置文件的事件的时候，会调用callback 🡨--- callback() epoll / select / poll

callback里干了一件事情，把这个socket添加到双向链表。

T： epoll，所有的soket都是由epoll管理的。

多个线程是不是是不是可以拥有不同的epoll？

T1：epoll\_fd1 – 1,2,3,4,5,6 master\_socket,

T2 : epoll\_fd2 – 11,12,13,14,15,16, master\_socket,

T3 : epoll\_fd3 – 21,22,23,24,25,26, master\_socket, REUSEPORT / REUSEADDR, 内核会对所有的线程排队，只会唤醒一个线程去处理。

监听端头：master\_socket 🡪 port

epoll -- > 多个socket